Une structure de découpe efficace pour l'affichage de grands modèles B-Rep

Frédéric Claux, David Vanderhaeghe, Loic Barthe, Mathias Paulin, Jean Pierre Jessel, David Croenne

Journées de l'Association Française d'Informatique Graphique, 2012

Une structure de découpe efficace pour l'affichage de grands modèles B-Rep

Frédéric Claux[1,2], David Vanderhaeghe[1], Loïc Barthe[1], Mathias Paulin[1], Jean-Pierre Jessel[1], David Croenne[1,2]

[1]IRIT - Université de Toulouse [2]Global Vision Systems

Résumé

Nous présentons une structure de découpe multirésolution pour l'affichage rapide et précis de modèles B-Rep. Nous proposons d'utiliser une représentation de la découpe de face basée sur un quadtree, autorisant une gestion efficace par le GPU. Ce quadtree contient des références à des courbes de découpe qui sont utilisées dans un fragment shader pour effectuer la classification de point d'une manière implicite. La façon dont sont stockées les informations multirésolution dans le quadtree nous permet de réduire les accès à notre structure lorsque les fragments sont distants de la caméra. Pour les objets en avant plan, nous nous appuyons sur la tessellation matérielle pour améliorer les performances, en réduisant la quantité de calcul à effectuer pour chaque fragment. Nous obtenons un affichage intéractif pour de très gros modèles comprenant des centaines de milliers de faces B-Rep, quel que soit le niveau de zoom.

1. Introduction

Le format B-Rep (B-Rep signifiant Boundary Representation en anglais, ou Représentation par Frontière) est la représentation la plus courante de la géométrie exportée par les applications de CAO (Conception Assistée par Ordinateur), où chaque face est décrite par une *surface support* tel qu'un plan, un cône ou une NURBS et des *courbes de découpe* définies dans l'espace *paramétrique* de chaque surface support. Chaque face a une courbe de découpe extérieure et, optionnellement, une ou plusieurs courbes de découpe intérieures définissant des trous, parfois avec des îles dedans. Les modèles CAO contiennent parfois des faces avec de grandes quantités de trous. Les courbes de découpe sont définies sous forme paramétrique avec des lignes, cercles, ellipses ou encore des courbes polynomiales ou rationnelles. Visualiser les modèles CAO d'une manière précise et en temps-réel est important dans l'industrie, mais demeure un problème scientifique, notamment pour les grands modèles.

La visualisation de modèles CAO, qui nous intéresse dans ce papier, doit idéalement être effectuée avec une précision illimitée (Figure 2). Les utilisateurs se déplaçant interactivement dans le modèle, à divers niveaux de zoom, souhaitent bénéficier d'un rendu très précis avec des courbes lisses, des chanfreins parfaitement arrondis, des formes et des trous parfaitement ronds, comme lorsqu'ils observent les pièces mécaniques une fois usinées.

Avant d'être donnés à un moteur de rendu en vue d'être affichés, les modèles B-Rep sont généralement tesselés, c'est-à-dire transformés en maillages, facilement affichés par les processeurs graphiques ou *GPUs* (Graphics Processing Units). Un inconvénient majeur de ce procédé est que cette transformation en maillage est effectuée à une résolution fixe. Plus cette tessellation est précise, plus le maillage est fin, plus l'affichage est fidèle mais aussi plus les performances sont faibles à cause du nombre accru de triangles. Certaines faces B-Rep contiennent parfois des centaines de courbes de découpe et la génération du maillage entraîne la création d'un très grand nombre de points et de triangles pour reproduire fidèlement les trous et contours définis par ces courbes. Il en résulte une occupation mémoire importante et de faibles performances à l'affichage. Des exemples de faces tesselées apparaissent en Figure 1.

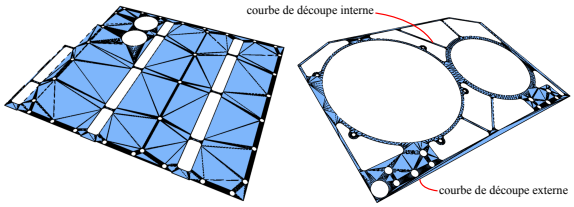


Figure 1: *La tessellation d'une face B-Rep définie avec beaucoup de courbes de découpe produit parfois de très nombreux triangles.*

Comme il a été montré dans des travaux précédents, il est possible d'utiliser les GPUs pour afficher les faces pendant le rendu, à partir d'une représentation adaptée, en effectuant des calculs à la volée. Une face B-Rep est représentée par une surface support et une structure de découpe, stockées en mémoire locale dans une texture. La structure de dé-

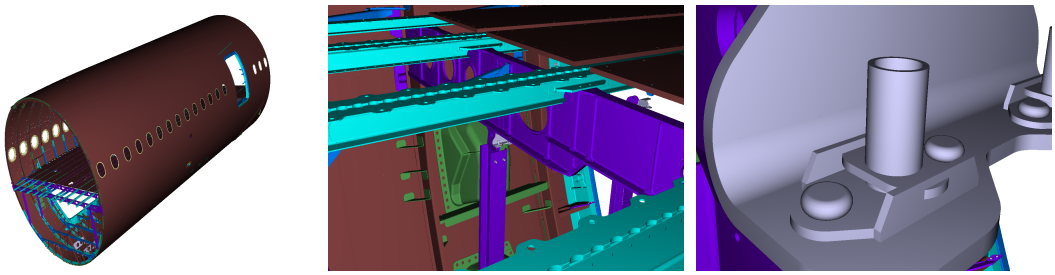


Figure 2: *Notre méthode de classification de point pour la gestion de la découpe pendant le rendu des surfaces B-Rep s'adapte bien au niveau de zoom. Quand les surface sont distantes de la caméra, un mécanisme multirésolution réduit le coût de la classification. Lorsqu'elles sont proches, nous nous appuyons sur la tessellation matérielle et passons par la classification de triangles entiers, lorsque c'est possible, pour accélérer l'affichage.*

coupe permet de déterminer si des points sur la surface de base sont effectivement dans ou hors de la zone de découpe. Ce procédé s'appelle la *classification de point*. On définit ces points comme étant *sur la face* ou *hors de la face*. La surface de base est rendue en utilisant la tessellation dynamique [SS09] ou le lancé de rayon [TL08], à la suite de quoi, pendant la rasterisation, les fragments *sur la face* et *hors de la face* sont classifiés grâce à une requête sur la structure de découpe. Seuls les fragments *sur la face* sont effectivement affichés. Bien qu'élégantes, ces méthodes d'affichage (voir l'état de l'art en Section 2) n'exhibent que de modestes performances pendant le rendu ou demandent une quantité de mémoire trop grande pour être utilisées pour la visualisation de grands modèles.

Contributions : Nous présentons une méthode permettant de visualiser de grands modèles B-Rep avec une précision visuelle élevée, et à une cadence d'affichage autorisant l'intéraction voire le temps réel. Notre contribution principale réside dans notre structure de découpe avec laquelle nous effectuons la classification de points. Nous proposons de représenter la découpe de chaque face par une structure vectorielle, multirésolution par nature. Nous avons observé (Section 2) que le recours au lancé de rayon pour la gestion des surfaces support est trop peu performant. L'utilisation de la tessellation introduit quasi inévitablement des cracks entre les faces (Figure 3). Fort de cette constatation, nous nous démarquons d'autres méthodes en représentant la découpe d'une manière délibérément approximée par rapport à la définition originale. Cette approximation produit quelques cracks supplémentaires entre les faces, à très fort niveau de zoom, mais permet de gagner en performances. La taille maximale autorisée de ces cracks est paramétrable lors d'un processus de prétraitement. L'aspect multirésolution de notre structure nous permet par ailleurs de réduire les calculs pour le traitement des fragments distants. Nous proposons enfin de nous appuyer sur la tessellation matérielle pour classifier rapidement des triangles entiers, notre structure étant bien adaptée au rendu basé sur la tessellation. Elle est non équilibrée par nature ce qui réduit l'occupation mémoire, notamment en comparaison avec les représentations basées sur des mipmaps.

Vue d'ensemble : Idéalement, la classification de point doit se faire en utilisant les courbes de découpe originelles,

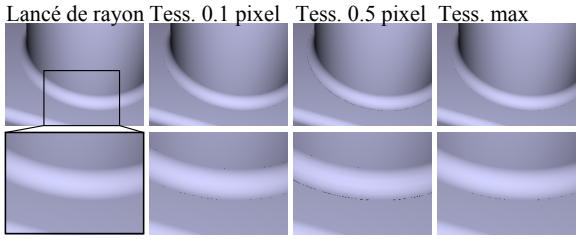


Figure 3: *Cracks entre deux faces causés par la tessellation, au niveau de la jonction d'un plan et d'un tore. La découpe circulaire au niveau du plan est effectuée ici avec la méthode de Schollmeyer et Fröhlich et ne souffre pas de problème de précision. Seule une gestion des surfaces support avec un lancé de rayon (à gauche) permet un résultat irréprochable, mais très lent à l'exécution. La tessellation du tore conditionnée à une erreur de corde de 0.1 pixel écran fait apparaître quelques cracks. A droite : pousser la tessellation à la valeur maximale supportée par le matériel – ce qui produit par ailleurs un nombre extrêmement élevé de triangles et réduit les performances – n'est pas suffisant pour supprimer toutes les imperfections.*

telles qu'elles peuvent être définies dans le modèle B-Rep issu de l'exportation. L'utilisation de telles courbes requiert cependant une grande quantité de calcul au moment du rendu. Plutôt que d'utiliser les courbes d'origine, nous approximons toutes les courbes d'entrée quel que soit leur type, avec une erreur configurable, par un ensemble de courbes de Bézier quadratiques connectées, représentables d'une manière implicite (Section 3), et stockées dans un quadtree couvrant l'espace paramétrique de chaque surface support (Section 4). Chaque nœud du quadtree contient une valeur de couverture binaire définissant si le nœud identifie une zone de l'espace paramétrique majoritairement *sur* ou *hors* de la face. Les feuilles de l'arbre qui ne sont pas complètement *sur* ou *hors* de la face contiennent aussi des références aux courbes d'approximation quadratiques les traversant. Cette structure, gérée d'une manière efficace par le GPU (Section 7), nous permet d'économiser de l'espace mémoire en comparaison avec les méthodes exposées lors de précédent travaux, d'autoriser un accès multirésolution réduisant le scintillement des fragments à l'écran et d'ac-

célérer leur classification pour les objets très distants (Section 5) ou très proches (Section 6).

Notre structure de données est seulement liée aux coordonnées *uv* du domaine paramétrique des surfaces support, que nous appellerons *l'espace paramétrique*. Bien qu'indépendante de la méthode de rendu utilisée, elle montre tous ses bénéfices lorsqu'elle est utilisée avec la tessellation matérielle des surfaces support. Pour chaque fragment rendu, nous calculons ses coordonnées *uv* en espace paramétrique et accédons à notre structure pour faire la classification. Pendant la traversée du quadtree, nous interrompons le parcours dès que nous atteignons un nœud couvrant moins d'un pixel, auquel cas nous utilisons la valeur de couverture présente dans le nœud. Si nous atteignons une feuille, si cette feuille couvre plus qu'un fragment à l'écran et fait référence à des courbes de découpe, nous effectons la ou les évaluation(s) quadratique(s) nécessaire(s) (Section 5.2). Pour les vues rapprochées, nous classifions les triangles issus de la tessellation (Section 6).

Avec notre méthode, les détails du modèle sont visualisés avec une fidélité supérieure à celle généralement obtenue avec des affichages utilisant la tessellation statique. Nous l'avons expérimentée avec des modèles industriels contenant des centaines de milliers de faces B-Rep, en obtenant toujours des temps de réponse compatibles avec l'intéractivité.

2. Etat de l'art

A chaque fragment rendu correspond un point qui doit être classifié comme étant *sur la face* ou *hors de la face*. Le concept des textures de découpe pour classifier les points avec des modèles à base de surfaces de type NURBS a été introduit par Guthe et al. [GBK05]. Guthe et al. transforment leurs surfaces de type NURBS et T-Splines en une hiérarchie de patchs de Bézier cubiques et de textures de découpe, où chaque nœud de la hiérarchie correspond à une erreur d'approximation fixée. En fonction du point de vue, des éléments de la hiérarchie sont sélectivement envoyés à la volée au GPU. La méthode de Guthe et al. requiert un streaming continu de données et montre ses limitations avec les plans très rapprochés, où la résolution des textures explose.

Pour apporter une réponse à ces problèmes de résolution, Schollmeyer et al. [SF09] définissent une stratégie de découpe dénuée de toute approximation. Ils décomposent d'abord les courbes de découpe du modèle exporté en courbes de Bézier rationnelles, puis en sections bimonotones. Ce travail préalable effectué après l'export leur permet de classifier les points d'une manière itérative pour effectuer la découpe pendant le rendu, avec des performances assez bonnes. Courbes et sections sont référencées dans une structure d'accélération basée sur un double arbre binaire. Leur méthode est très précise puisqu'ils n'ont recours à aucune approximation. Lorsque les surfaces support sont rendues avec un lancé de rayon, la précision de l'affichage est quasiment irréprochable. Les performances diminuent toutefois lorsque trop de fragments doivent être classifiés, en lancé de rayon comme en tessellation.

La nature vectorielle des structures de découpe partage des idées communes avec le concept de textures vectorielles et de rendu basé chemin (*path rendering*, en anglais). Nehab et Hoppe [NH08] font du rendu basé chemin en temps réel. Leur approche est de pré-construire une grille régulière de résolution donnée sur un dessin vectoriel, et de restreindre la problématique de rendu basé chemin à chaque cellule de la grille. Leur approche n'a pas de niveaux de détails. La taille de leur grille est fixée manuellement avec un compromis entre les performances à la l'affichage et la consommation mémoire.

Ray et al. [RNCL05] proposent le concept des Vector Texture Maps, ou VTM. Une VTM est une structure vectorielle multirésolution. Elle utilise des courbes vectorielles implicites et monotones restreintes à des cellules d'un arbre de subdivision. Les cellules ne peuvent contenir que une ou deux courbes vectorielles devant être représentées par une fonction polynomiale, matérialisant une approximation des courbes originelles, sinon une subdivision 4x4 est appliquée à la cellule. Les courbes générées sont propres à chaque cellule. La classification de point est faite en évaluant l'unique courbe trouvée dans une cellule, ou les deux courbes puis en combinant le résultat logiquement. Leur structure multirésolution est basée sur un mipmap : ils construisent une pyramide mipmap avec une résolution maximale correspondant à la plus petite cellule obtenue lors de la subdivision. Les courbes de découpe que l'on trouve dans les modèles CAO peuvent entraîner la création de quadtrees d'une profondeur de 12 ou plus, ce qui nécessiterait la création d'une texture de dimension 2^{12}. Cette représentation gourmande en mémoire n'est pas adaptée à nos modèles, qui comportent des millions de faces.

Hanniel and Haller [HH11] ont une approche originale pour effectuer la découpe à la volée. Leur méthode de rendu ne se base pas sur la représentation explicite de zones de découpe définies dans l'espace paramétrique de chaque surface support. Ils proposent un algorithme de classification de fragment basé sur un lancé de rayon en deux passes successives. Un lancé de rayon est tout d'abord effectué sur la surface support considérée pour le rendu. Une structure est utilisée pour déterminer la face adjacente à la face courante, proche du fragment en cours de traitement. Si une face adjacente proche est trouvée, un lancé de rayon est effectué contre la surface support correspondante. Ils déduisent la classification en comparant la distance des deux fragments obtenus. Leur méthode n'est pas assez performante pour être utilisée pour de grands modèles et ne marche qu'avec le lancé de rayon. Enfin, l'information d'adjacence entre les faces n'est pas toujours disponible dans les modèles B-Rep – certains formats ne décrivent que les surfaces support et leurs courbes de découpe, sans information de connectivité.

3. Approximation des courbes de découpe

La première étape dans notre approche, faite dans un prétraitement, est de générer une représentation des courbes de découpe permettant une exploitation rapide par le GPU.

3.1. Approximation avec des courbes de Bézier quadratiques connectées

Les courbes de découpe du modèle B-Rep d'entrée sont tout d'abord approximées avec des courbes de Bézier

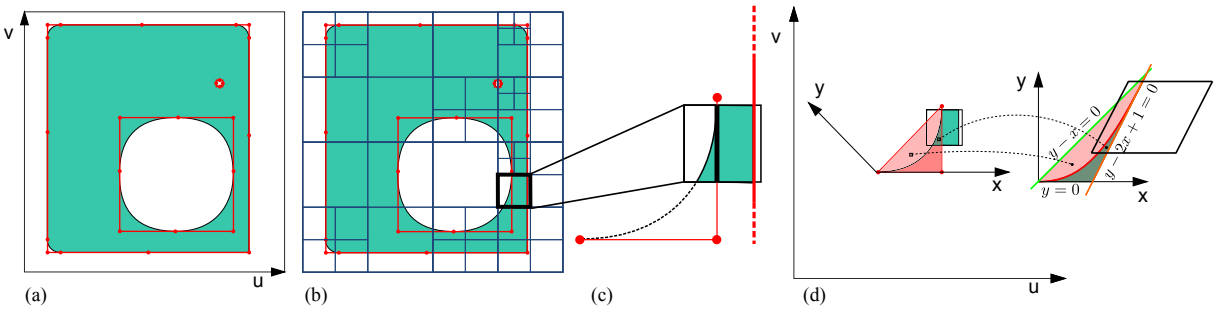


Figure 4: *(a) Approximation quadratique des courbes de découpe. Les points et polygones de contrôle sont affichés en rouge. (b) Quadtree : chaque cellule peut recouvrir une zone complètement sur la face, hors de la face, ou partiellement sur et hors, avec une ou deux courbes quadratiques qui traversent la cellule. (c) Lorsqu'une feuille a deux courbes quadratiques qui la traverse, les portions de courbes intersectantes sont isolées par une ligne séparatrice qui est stockée dans la cellule. (d) Pour évaluer une courbe quadratique d'une manière implicite, nous transformons ses coordonnées paramétriques (u,v) en coordonnées quadratiques (x,y).*

quadratiques, en respectant une erreur d'approximation ε, exprimée en espace monde, comme illustré sur la Figure 4. Nous discrétisons chaque courbe d'entrée et effectuons un *fitting* produisant un Bspline uniforme de degré 2, interpolant les points discrétisés. Chaque Bspline est ensuite simplifié avec un algorithme de suppression de nœud guidé par une erreur d'approximation exprimée en espace paramétrique, calculée à partir de ε, et décomposé en courbes de Bézier quadratiques connectées entre elles [LM87, LM88]. Une continuité C^1 (dérivées premières tangentes) est assurée d'une courbe à l'autre, lorsque les courbes d'entrée sont elles-mêmes continues C^1. Les courbes de Bézier quadratiques plates sont converties en segments de ligne.

3.2. Reconstruction implicite locale

En suivant la convention utilisée pour les courbes de découpe d'entrée, nos courbes quadratiques sont toujours orientées de telle sorte que la partie *sur la face* se situe toujours à gauche de la courbe. La classification de point consiste à déterminer de quel côté de la courbe un point se situe. A l'intérieur de l'enveloppe convexe du polygone de contrôle de la courbe quadratique, nous effectuons la classification de point avec la méthode introduite par Loop et Blinn [LB05].

Loop et Blinn utilisent une fonction $f : \mathbb{R}^2 \to \mathbb{R}$, définie par $f(x,y) = y - x^2$ si la face est localement convexe et $f(x,y) = x^2 - y$ si la face est localement concave, pour représenter une courbe de Bézier quadratique, où (x,y) est exprimé dans l'*espace quadratique* défini par un *repère quadratique* (Figure 5). Dans ce repère, la courbe quadratique est l'ensemble des valeurs nulles de f. Le repère est aussi défini de sorte que, les points de contrôle successifs p_i, $i = 0,1,2$ de la courbe de quadratique aient des coordonnées (x_i,y_i) successivement égales à $(0,0)$, $(0.5,0)$ et $(1,1)$ (Figure 5). Dans l'enceinte de l'enveloppe convexe du polygone de contrôle de la courbe quadratique, les coordonnées (x,y) évaluées en $f(x,y) \geq 0$ sont *sur la face*.

Pour classifier un point de coordonnées (u,v) dans l'espace paramétrique, ses coordonnées (x,y) en espace quadratique sont calculées comme suit :

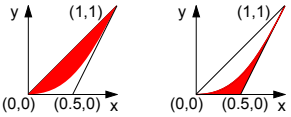


Figure 5: *Illustration de la partie sur la face (en rouge) définie par la représentation implicite de la courbe de Bézier dans l'espace quadratique. A gauche : une zone sur la face convexe vérifiant $y - x^2 \geq 0$. A droite : une zone sur la face concave vérifiant $x^2 - y \geq 0$.*

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2(u_{p_1} - u_{p_0}) & u_{p_0} - 2u_{p_1} + u_{p_2} & -u_{p_0} \\ 2(v_{p_1} - v_{p_0}) & v_{p_0} - 2v_{p_1} + v_{p_2} & -v_{p_0} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

où (u_{p_i}, v_{p_i}), $i = 0,1,2$ sont les coordonnées respectives des points de contrôle de Bézier dans l'espace paramétrique. Une fois que c'est fait, évaluer $f(x,y)$ nous dit si le point est *sur la face* ($f(x,y) \geq 0$) ou *hors de la face* ($f(x,y) < 0$).

4. Construction de la structure multirésolution

La représentation multirésolution de la découpe de la face s'appuie sur une subdivision récursive de l'espace paramétrique.

Les cellules feuilles de l'arbre de subdivision identifient une zone complètement *sur la face* ou *hors de la face*, ou bien une zone qui est traversée par deux courbes quadratiques au plus. La subdivision de cellule continue jusqu'à ce que chaque cellule respecte ces contraintes. Dans les feuilles sont stockées jusqu'à deux références de courbes quadratiques intersectantes. Ceci est illustré dans la Figure 4(b).

La présence de deux références à des courbes quadratiques dans une cellule est inévitable, les points de jonction entre des courbes quadratiques adjacentes n'ayant quasiment aucune chance de tomber sur des frontières de cellule. Par ailleurs, deux courbes de découpe distinctes peuvent parfois

se trouver tellement proches l'une de l'autre qu'une division récursive de la cellule doive être faite d'une manière très profonde avant que les contraintes de cellule ne soient respectées. La Figure 6 montre un quadtree construit sur le domaine paramétrique pour deux faces B-Rep complexes.

La subdivision récursive descendante des cellules est suivie par un calcul de la valeur de couverture, obtenue en remontant le quadtree, de bas en haut. Cette valeur de couverture, binaire, est utilisée pendant le rendu comme mécanisme multirésolution (Section 5.2).

5. Classification des points dans les cellules

La classification de points pour les cellules complètement *sur la face* ou *hors de la face* est triviale. Nous développons maintenant la classification pour les cellules disposant de courbes intersectantes.

5.1. Cellules avec une seule courbe quadratique

Dans une cellule traversée par une seule courbe convexe, les coordonnées paramétriques (u,v) d'un point sont converties en coordonnées quadratiques (x,y) comme expliqué en Section 3.2. On teste ensuite si le point se trouve dans le polygone de contrôle. Il est classifié comme étant *hors de la face* si $y<0$ ou $y-2x+1<0$, *sur de la face* si $y-x\geq 0$; en dehors du polygone de contrôle, on classifie le point avec une évaluation implicite de la fonction f présentée en Section 3.2 (Figure 4(d)). Dans le cas concave, les classifications *sur la face* et *hors de la face* sont inversées.

5.2. Cellules avec deux courbes quadratiques

Lorsque deux courbes quadratiques traversent une cellule, nous précalculons une ligne séparatrice. En considérant seulement les portions de courbe qui traversent effectivement les cellules, une ligne séparatrice est une ligne qui ne coupe aucune des deux courbes quadratiques, avec une courbe de chaque côté (see Figure 4(c)) ; elle peut être construite à l'aide du théorème de l'axe de séparation [Ebe06, p. 393] : étant donnés deux polygones convexes disjoints, il existe toujours une ligne avec un polygone de chaque côté. De plus, il existe un segment d'un des deux polygones qui peut servir de support à cette ligne de séparation (Figure 4(c)). Un tel segment peut être trouvé en subdivision les polygones de contrôle jusqu'à ce qu'ils ne soient pas sécants entre eux. Cela est toujours possible puisque les faces sont *2D-Manifold* et que les courbes de Bézier ne se croisent jamais. Les lignes construites à partir des segments des polygones de contrôles, progressivement subdivisés (par exemple avec le schéma de De Casteljau) sont testées en vérifiant que l'autre polygone de contrôle n'est pas coupé. La première ligne qui convient est choisie comme ligne de séparation pour la cellule.

Pendant le rendu nous testons d'abord de quel côté de la ligne un point se situe, puis nous procédons à la classification de point décrite en Section 5.1.

5.3. Mécanisme d'accès multirésolution

La valeur de couverture, binaire, détermine si une cellule occupe une zone de l'espace paramétrique majoritairement *sur la face* ou *hors de la face*. Les cellules feuilles qui sont complètement *sur* ou *hors* de la face sont identifiées dans le quadtree par une valeur de couverture égale à 1 ou 0, respectivement, sans référence à des courbes quadratiques.

Pendant le rendu, le traversée du quadtree est arrêtée quand un nœud recrouve moins d'un pixel à l'écran. Lorsqu'une feuille de l'arbre est atteinte et que son empreinte à l'écran recouvre plusieurs pixels, la classification de point est effectuée suivant la procédure décrite dans les Sections 5.1 ou 5.2. Cette méthode d'accès multirésolution nous permet d'augmenter les performances à l'affichage en limitant la profondeur d'accès au quadtree (Figure 7 et 8(a)) et de limiter l'effet de moiré visuellement inélégant (Figure 8(b)) sur certaines faces distantes.

6. Classification de triangle

La structure B-Rep d'un modèle exporté en CAO permet d'avoir un indice sur la façon dont les différentes composantes ont été conçues. Des faces B-Rep de très grande taille, mais ne contenant que quelques petites zones éparses aussi bien *sur la face* que *hors de la face*, sont communes, en dépit de la taille parfois très grande des surfaces support. Ces faces sont enclines à une dégradation des performances puisqu'elles mènent au traitement d'un grand nombre de fragments, qui pourraient idéalement être classifiés de manière relativement triviale, aussi bien *sur la face* qu'*hors de la face*. Voir le modèle d'avion de la Figure 9.

Lorsque les faces B-Rep sont proches de la caméra, les

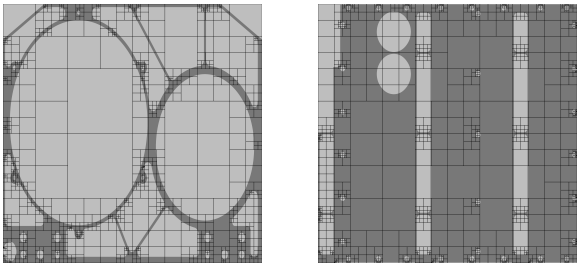


Figure 6: *Quadtrees générés pour les faces présentées en Figure 1*

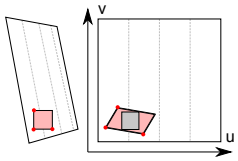


Figure 7: *A gauche : empreinte d'un fragment en espace écran. Le domaine paramétrique est déformé pour tenir compte de la projection et perspective de la vue. A droite : empreinte du même fragment en espace paramétrique. Nous calculons la taille du rectangle le plus grand qui tient dans l'empreinte pour limiter la profondeur de traversée du quadtree.*

performances peuvent diminuer d'une manière spectaculaire puisque la classification de point doit être faite pour un nombre très élevé de fragments : même les petites surfaces génèrent par ailleurs un grand nombre de fragments lorsqu'elles sont fortement zoomées.

Le coût de la classification de point dans notre méthode vient principalement de la traversée du quadtree, un coût qui est proportionnel à la profondeur de la cellule à utiliser. La classification avec les courbes est relativement rapide, puisqu'elle utilise des évaluations implicites. Pour réduire la quantité de calculs et d'accès mémoire à effectuer, nous proposons de grouper la classification de fragments et de faire la classification des triangles issus de la tessellation dynamique. Pour les triangles que nous ne pouvons pas classifier d'un bloc, dans leur intégralité, nous accélérons le temps de classification des fragments sous-jacents.

Les surfaces support sont tesselées sur leur domaine paramétrique u, v pendant le rendu, en respectant une erreur écran (voir Section 7). Pour chaque surface, nous prédéterminons un profondeur (ou niveau) de quadtree k_i à laquelle nous savons que nous pourrons classifier pendant le rendu au moins 40% du domaine paramétrique quand ce dernier est couvert de triangles issus de la tessellation, avec des facteurs valant $t_u = 2^{k_i}, t_v = 2^{k_i}$. Pendant le rendu, nous voulons que les triangles tesselés respectent à la fois l'erreur écran que nous nous sommes fixée, et qu'ils soient alignés sur les frontières des cellules du quadtree, au moins jusqu'au niveau k_i, garantissant alors qu'au moins 40% d'entre eux seront rapidement classifiés pendant le rendu. Nous calculons donc pendant le rendu des facteurs de tessellation qui respectent l'erreur écran, puis nous adjustons ces facteurs à des puissances de deux supérieures, au moins égales à 2^{k_i}, ce qui nous donne des facteurs finaux $2^{k_{uf}}, 2^{k_{vf}}$. La Figure 10 montre comment nous ajustons les facteurs de tessellation de sorte que suffisamment de triangles soient classifiés.

Pour chaque triangle tesselé, nous procédons à la classification avant que la rasterisation ne prenne place, comme montré en Figure 11. Une seule cellule est identifiée dans le quadtree jusqu'à un niveau de $k_f = \min(k_{uf}, k_{vf})$, une fois par triangle, en utilisant son barycentre. A ce stade, un triangle peut être classifié comme étant *sur* ou *hors* de la face si la cellule est une cellule feuille qui

- identifie une zone complètement *sur la face* ou *hors de la face*,
- référence une courbe quadratique, et le triangle réside du côté convexe de cette courbe,
- référence deux courbes quadratiques, et les points du triangle résident tous du même côté de la ligne de séparation, et du côté convexe de la courbe quadratique correspondante.

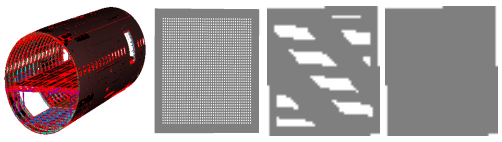
Autrement, lorsque le triangle ne peut être classifié, la référence au nœud du quadtree est passée au fragment shader. La classification des fragments reprendra a partir de ce nœud, et non de la racine du quadtree.

Le gain de performance obtenu avec la classification de triangle doit être supérieur à la diminution de performance liée à la tessellation additionnelle effectuée, une tessellation qui serait normalement effectuée avec des facteurs respectant seulement l'erreur écran que nous nous sommes fixée. La classification de triangle est proportionnellement intéressante à mesure que le nombre moyen de fragments par triangle augmente. En moyenne, le coût de la classification de n fragments sur un triangle T, considéré comme une zone de l'espace écran, doit être inférieur lorsque la tessellation a été réalisée, que lorsqu'elle n'a pas été faite, i.e.

$$C^{T(k_{uf},k_{vf})} + \sum_{i=1}^{n} C_f^{T(k_{uf},k_{vf})}(i) < \sum_{i=1}^{n} C_f(i)$$

où $C^{T(k_{uf},k_{vf})}$ représente le coût de la classification de triangle, C_f le coût de la classification des fragments contenus dans l'espace écran de ce triangle et $C_f^{T(k_{uf},k_{vf})}$ le coût de cette même classification en prenant en compte les calculs supplémentaires effectués pour le triangle, réutilisés pour chaque fragment du triangle.

Nous avons testé la classification de triangle avec des facteurs de tessellation uniformes $t_u = t_v$ pour des surfaces planes, étant donnée leur très grande quantité dans les modèles CAO (voir Table 1), vu que le facteur de tessellation n'a pas d'impact sur l'erreur écran et parce que nous pouvons calculer le nombre de fragments tôt dans le pipeline de rendu pour ces surfaces (eg. shader de contrôle des facteurs de tessellation). Nous activons notre méthode lorsque les tri-



(a) (b)

Figure 8: ***(a)*** *Les fragments rouges tirent bénéfice de l'accès multirésolution. Ils identifient des fragments pour lesquels la traversée du quadtree s'est arrêtée à un nœud intermédiaire, ou à un nœud feuille mais où l'utilisation des courbes quadratiques n'a pas été nécessaire.* ***(b)*** *A gauche : exemple de face B-Rep avec de nombreux petits trous. Au centre : moiré visible à faible niveau de zoom. A droite : en utilisant l'accès multirésolution, les trous disparaissent à faible niveau de zoom, mais il n'y a pas de moiré.*

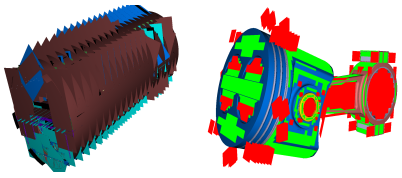


Figure 9: *A gauche : modèle d'avion rendu sans la découpe des faces activée, révélant de très nombreuses surfaces planes dans le sens transversal du fuselage. Le fuselage a beaucoup de sections circulaires qui ont été conçues avec un seul objet dans le modèle CAO. A droite : les zones vertes et rouges dans le modèle du piston correspondent à des triangles pour lesquels la classification a été réalisée avec succès. Les triangles rouges sont éliminés dans le shader géométrique, avant que la rasterisation ne prenne place.*

angles couvrent au moins 200 pixels à l'écran. C'est assez pour augmenter considérablement les performances. Par exemple, la Vue 1 du modèle de Piston tourne deux fois plus vite avec la classification de triangle que sans.

7. Implémentation

Les modèles CAO comportent des faces ayant pour support des surfaces de nombreux types. Comme expliqué par Toledo et Levy [TL08], le lancé de rayon est bien adapté à la visualisation de ces modèles. Cette méthode de rendu produit un résultat de grande qualité avec des performances comparables à (voire dépassant) la tessellation pour plusieurs types de surfaces support. Néanmoins, le lancé de rayon reste très lourd pour les surfaces plus complexes. Notre méthode de découpe étant dans ses grandes lignes indépendante de l'algorithme de rendu des surfaces support, notre moteur graphique peut aussi bien tesseler à la volée toutes les surfaces support ou passer par un lancé de rayon pour les surfaces simples. Nous avons implémenté le lancé de rayon pour les sphères, cylindres et cones ; les tores, fillets et patchs de Bézier cubiques utilisent des shaders de tessellation dédiés. Les surfaces d'autres types sont converties en patchs de Bézier cubiques dans un prétraitement [LM87, LM88], moyennant une erreur définie en espace monde.

Lorsque la tessellation est utilisée, chaque surface support est passée aux shaders de tessellation sous forme d'un seul patch contenant essentiellement des références à des emplacements en mémoire vidéo où la définition de la surface est stockée. Deux facteurs de tessellation sont utilisés, un pour chaque dimension du domaine paramétrique. Le facteur de tessellation supporté par le matériel actuel étant limité à 64, ce qui signifie que 4096 sommets seront créés, au plus, pour un seul patch de surface support. Notre stratégie de tessellation est dictée par une erreur en espace écran ε_s, exprimée en pixels, définissant la déviation maximale autorisée entre la surface support telle qu'elle est géométriquement définie et sa discrétisation, ainsi que par les facteurs de tessellation t_u et t_v (Section 6).

8. Résultats

Nous utilisons trois modèles CATIA V5 pour nos tests, un piston, un satellite et une section d'avion.

Qualité visuelle : Notre méthode de rendu fournit une précision visuelle élevée dans les plans rapprochés. Alors que la tessellation statique montre ses limitations en terme de performances et d'occupation mémoire quand elle est conjointement utilisée pour l'affichage de grands modèles et pour visualiser des détails d'une grande finesse, notre approche permet d'afficher des courbes de découpe d'une manière lisse (Figure 12).

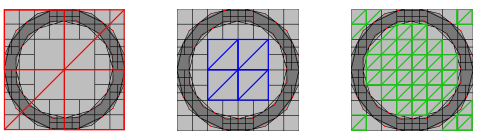


Figure 10: *Les triangles rouges alignés sur le niveau 1 du quadtree ne peuvent être classifiés, alors que les triangles de niveau 2 et 3 = k_i affichés en bleu et vert le peuvent, couvrant respectivement 25% et 50% du domaine paramétrique.*

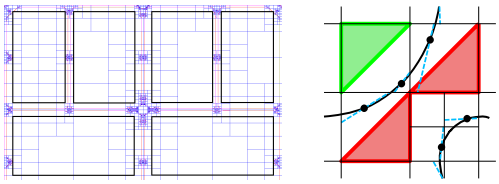


Figure 11: *A gauche : surface plane avec de grandes zones hors de la face affichées en noir. A droite : classification de triangle. La classification des triangles rouges échoue puisque leurs sommets sont soit des deux côtés de la ligne séparatrice de la cellule (à droite), ou du côté concave de la courbe (en bas). Les triangles verts pour qui la classification a pu être effectuée seront soit rapidement classifiés en tant que zones sur la face, ou éliminés par le shader géométrique lorsqu'ils sont hors de la face.*

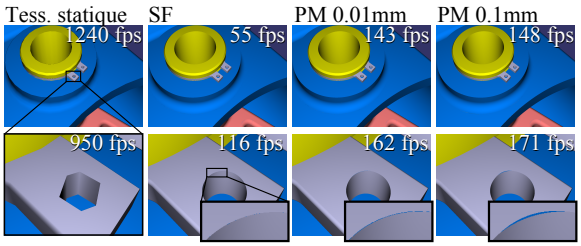


Figure 12: *Examen de la qualité du rendu pour le modèle de piston. Tess. statique : tessellation statique de 0.1mm, SF : Schollmeyer et Fröhlich, PM : méthode proposée avec $\varepsilon = 0.01mm$ ou $0.1mm$. Rendu effectué en tessellation dynamique avec une erreur d'écran de $\varepsilon_s = 0.1$. Temps de rendu fourni de manière indicative, exprimé en images par seconde (fps, de l'anglais frames per second).*

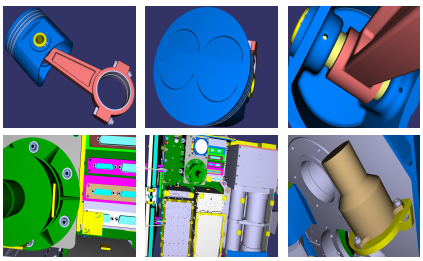


Figure 13: *Vues utilisées pour l'évaluation de la performance de rendu pour le modèle de Piston (en haut) et celui du Satellite (en bas).*

Performance : La Table 2 montre les résultats des tests de performance obtenus avec notre méthode et celle de Schollmeyer et al. La tessellation est effectuée avec l'API Datakit pour la gestion des modèles CATIA (`www.datakit.com`). Dans ce test, nous fixons la tolérance à 0.1 mm pour les surfaces de Bézier et pour les courbes sup-

Type de surface support	Satellite	Avion	Piston
Plans	36518	276788	210
Cylindres	41832	295262	390
Cônes	9722	20488	26
Sphères	667	1595	24
Tores	6517	34266	66
Fillets	166	22125	74
Nurbs	853	20871	0
Autres types	173	30669	0
Total	96448	702064	790
Patchs de Bézier cubiques	1271	71894	0
Nombre total de patchs	96693	722418	790

	Courbes de découpe		
Type de courbe	Satellite	Avion	Piston
Ligne	354197	2767813	2680
Arc d'ellipse	58797	438295	294
Nurbs (total)	58236	496808	606
Nurbs de degré 1	12479	2251	0
Nurbs de degrés 2,3,4	98	407	8
Nurbs de degrés 5	45571	493723	598
Nurbs de degrés 6,7,8,11	88	427	0
Nombre total de courbes	471230	3702916	4186
Quadratiques (0.1mm)	690140	5975794	5695
Quadratiques (0.01mm)	941476	8661440	7854

Table 1: *Propriétés du modèle classées par type de surface support et de courbe de découpe. La ligne Bézier cubiques compte le nombre de patchs de Bézier créés après le processus d'approximation des surfaces support originales. Toute surface support qui n'est pas d'un type explicitement pris en charge (plan, cône, cylindre, tore, sphere ou fillet) est approximée en un ou plusieurs patch(s) de Bézier cubique(s). Autrement, à chaque surface support correspond un patch sur le GPU.*

port des fillets. Nous gérons l'approximation des courbes de découpe de sorte que ces dernières ne dévient pas de plus de 0.1 ou 0.01 mm, en espace monde, des courbes d'origine. Les résultats sont obtenus avec une GeForce GTS 450 avec 1 Go de mémoire vidéo, sur un processeur Intel Core i7-860 avec 8 Go de RAM. Notre méthode permet d'obtenir un gain de performance allant de 10% à 240% suivant le modèle et la vue utilisés.

Occupation mémoire : Les modèles statiquement tesselés occupent un espace mémoire qui augmente proportionnellement avec les facteurs de discrétisation. Pour les données qui servent à définir les surfaces support, l'occupation mémoire de la méthode proposée est affectée par la tolérance à l'erreur utilisée pour approximer les surfaces complexes avec des patchs de Bézier cubiques, et les courbes support pour les fillets, approximées avec des courbes de Bézier cubiques. L'occupation mémoire de la structure de découpe est affectée par l'erreur d'approximation utilisée pour les courbes de découpe. La Table 3 montre l'occupation mémoire pour notre méthode et celle de Schollmeyer et al., ainsi que pour des modèles statiquement tesselés. Avec une erreur d'approximation des courbes de découpe de ε = 0.1 mm, nous consommons 30-45% moins de mémoire que Schollmeyer et al. Avec ε = 0.01 mm, nous obtenons à peu la même occupation.

Limitations : Dans certaines circonstances rares, le processus de subdivision des cellules peut dégénérer quand trois courbes quadratiques ou plus sont très rapprochées,

	Temps de rendu (ms)					
	(a)	(b)	(a)	(b)	(a)	(b)
Avion	Vue 1		Vue 2		Vue 3	
SF	145	130	115	101	50	56
P.M. 0.1mm	108	98	103	92	38	37
P.M. 0.01mm	109	99	93	92	40	37
Satellite	Vue 1		Vue 2		Vue 3	
SF	44	28	45	31	87	22
P.M. 0.1mm	34	21	30	21	65	14
P.M. 0.01mm	33	20	30	19	65	15
Piston	Vue 1		Vue 2		Vue 3	
SF	15	8.3	31	15	53	25
P.M. 0.1mm	8.8	3.1	19	4.5	31	8
P.M. 0.01mm	8.9	3.1	19	4.6	33	8

Table 2: *Temps de rendu en millisecondes pour notre méthode (P.M.) et celle de Schollmeyer et Fröhlich's (SF). Les vues utilisées sont celles montrées en Figures 2 et 13. Approximation de surface ε = 0.1mm ; Erreur de tessellation écran ε_s = 0.5 pixel. Résultats présentés aussi bien en lancé de rayon + tessellation **(a)** ou en tessellation seulement **(b)***

.

Section d'avion	Surface	Découpe	Total
SF	64	296	360
P.M. 0.1mm	64	197	261
P.M. 0.01mm	64	294	358
Discrétisation 0.1mm			246
Discrétisation 0.01mm			1010
Satellite	Surface	Découpe	Total
SF	10.5	48.1	58.6
P.M. 0.1mm	10.5	31	41.5
P.M. 0.01mm	10.5	43.7	54.2
Discrétisation 0.1mm			42.5
Discrétisation 0.01mm			139.3
Piston	Surface	Découpe	Total
SF	0.119	0.656	0.775
P.M. 0.1mm	0.119	0.309	0.428
P.M. 0.01mm	0.119	0.440	0.559
Discrétisation 0.1mm			0.629
Discrétisation 0.01mm			2.683

Table 3: *Occupation mémoire, en mégaoctets, pour la méthode de Schollmeyer et Fröhlich (SF), pour la méthode proposée (P.M.), et pour des modèles statiquement tesselés (pour lesquels chaque sommet prend 36 octets, en prenant en compte l'espace requis pour les coordonnées des sommets, leur normale et couleur). L'approximation des surfaces en patchs de Bézier cubiques, effectuée pour la prise en charge de surfaces complexes, utilise dans tous les cas une tolérance de 0.1 mm.*

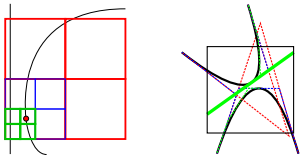


Figure 14: *A gauche : le cercle rouge identifie un point de jonction entre deux courbes quadratiques. La subdivision doit parfois être effectués profondément avant d'obtenir des cellules respectant nos contraintes (2 courbes quadratiques au maximum par cellule). A droite : un nœud du quadtree avec deux courbes traversantes. Une ligne séparatrice est construite après 3 subdivisions d'un des deux polygones de contrôle.*

ce qui se produit principalement lorsqu'une extrémité de courbe quadratique appartenant à une première boucle est très proche d'une seconde boucle (Figure 14, à gauche). Dans ce cas, la subdivision doit se faire très profondément dans le quadtree puisque l'extrémité de la courbe quadratique créée une situation multi-courbe que nous ne pouvons prendre en charge dans notre modèle de cellule, limité à deux courbes. Cette situation se produit le plus souvent parce que nous ne contrôlons pas les endroits où des points de jonction sont créés entre les courbes quadratiques, pendant le processus d'approximation. La Table 4 montre que cette situation est très rare cependant, même avec le modèle de section d'avion, contenant un très grand nombre de faces.

Une solution envisageable pour ce problème serait de décaler les points de contrôle des courbes quadratiques le long des courbes de découpe de sorte que ces derniers s'éloignent des boucles gênant la subdivision. Il est également possible de générer une nouvelle courbe quadratique, juste pour le segment traversant une cellule. Enfin, les cellules problématiques pourraient être subdivisées d'une manière différente qu'une simple subdivision 2x2 uniforme ; toutefois cette approche rendrait plus complexes le prétraitement comme le code utilisé pour le rendu.

La méthode que nous utilisons pour calculer la ligne séparatrice (Section 5.2) peut prendre un certain temps avant de converger vers une solution. Quand les segments support sont trop rapprochés, la subdivision des polygones de contrôle doit dans certains cas être assez poussée avant qu'un segment ne puisse être utilisé pour construire la ligne séparatrice (Figure 14 à droite).

9. Conclusion et travaux futurs

Nous avons présenté une méthode de rendu basée sur la découpe directe de surfaces B-Rep dynamiquement tesselées ou affichées via un lancé de rayon. La nature multirésolution de notre structure de découpe de type quadtree et l'existence d'une valeur de couverture dans les cellules nous permet de réduire le nombre d'accès à notre structure pendant le rendu lorsque les surfaces sont distantes de la caméra, ce qui améliore les performances notamment lorsqu'un grand nombre de fragments distants doivent être traités. Lorsque la tessellation est utilisée pour la gestion des surfaces support, nous proposons de nous appuyer sur l'unité de tessellation de la carte graphique, très performante, pour accélérer la classification d'un grand nombre de fragments, pour les objets rapprochés.

Section d'avion	0.1mm	0.01mm
Nombre de patchs	365	668
% du nombre de patchs total	0.12	0.22
Nœuds par patch (moyenne)	2.5	4.4
Satellite		
Number of patchs	147	333
% du nombre de patchs total	0.23	0.5
Nœuds par patch (moyenne)	1.2	3
Piston		
Number of patchs	0	0
% du nombre de patchs total	0	0
Nœuds par patch (moyenne)	0	0

Table 4: *Nombre de patchs pour lesquels la profondeur de subdivision du quadtree a atteint le niveau 13, entraînant un abandon (Voir Figure 14 à gauche). La table montre aussi le nombre moyen de nœud problématique trouvé en moyenne dans une structure de découpe où au moins un problème a été détecté.* 1.2 *signifie que, en moyenne, une structure de découpe problématique a* 1.2 *nœud problématique.*

Nous avons testé cette dernière méthode pour les surfaces planes. Nos travaux se poursuivent pour dégager un algorithme plus général pour traiter tous les types de surfaces support. En particulier, les cylindres, cones et autres surfaces typiquement tesselés en $[1,n]$ subdivisions peuvent voir leur performance réduite à l'affichage si l'algorithme est mal configuré, dans certaines circonstances. Nos tests montrent que le fait que le facteur de tessellation doive toujours être être aligné sur une puissance de deux n'est pas toujours profitable en terme de performances si nous ne pouvons pas estimer d'une manière fiable le nombre de fragments générés pour une primitive suffisamment tôt dans le pipeline de rendu. Une heuristique pourrait être trouvée pour déterminer les facteurs de tessellation optimaux pour une surface dans un contexte dépendant de la vue, de sorte que, globalement, la classification de la surface soit la plus efficace possible, en prenant en compte le nombre de fragments et la complexité effective du quadtree. Notre méthode peut également être appliquée avec de la tessellation par subdivision – le quadtree se prêtant fort bien à cet exercice. Il est également envisageable d'adapter l'algorithme de sorte que davantage de triangles puissent être rapidement classifiés, et non pas seulement ceux contenus dans l'enceinte d'un seul nœud du quadtree.

Références

[Ebe06] EBERLY D. H. : *3D Game Engine Design, Second Edition : A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[GBK05] GUTHE M., BALÁZS A., KLEIN R. : Gpu-based trimming and tessellation of nurbs and t-spline sur-

faces. *ACM Trans. Graph.. Vol. 24* (July 2005), 1016–1023.

[HH11] HANNIEL I., HALLER K. : Direct rendering of solid cad models on the gpu. In *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on* (septembre 2011), pp. 25 –32.

[LB05] LOOP C., BLINN J. : Resolution independent curve rendering using programmable graphics hardware. *ACM Trans. Graph.. Vol. 24* (July 2005), 1000–1009.

[LM87] LYCHE T., MØRKEN K. : Knot removal for parametric b-spline curves and surfaces. *Computer Aided Geometric Design. Vol. 4*, Num. 3 (1987), 217 – 230.

[LM88] LYCHE T., MØRKEN K. : A data-reduction strategy for splines with applications to the approximation of functions and data. *IMA Journal of Numerical Analysis. Vol. 8*, Num. 2 (1988), 185–208.

[NH08] NEHAB D., HOPPE H. : Random-access rendering of general vector graphics. *ACM Trans. Graph.. Vol. 27* (December 2008), 135 :1–135 :10.

[RNCL05] RAY N., NEIGER T., CAVIN X., LÉVY B. : *Vector Texture Maps*. Tech. rep., INRIA - ALICE, 2005.

[SF09] SCHOLLMEYER A., FRÖHLICH B. : Direct trimming of nurbs surfaces on the gpu. *ACM Trans. Graph.. Vol. 28* (July 2009), 47 :1–47 :9.

[SS09] SCHWARZ M., STAMMINGER M. : Fast gpu-based adaptive tessellation with cuda. *Computer Graphics Forum. Vol. 28*, Num. 2 (2009), 365–374.

[TL08] TOLEDO R., LEVY B. : Visualization of industrial structures with implicit gpu primitives. In *Proceedings of the 4th International Symposium on Advances in Visual Computing* (Berlin, Heidelberg, 2008), ISVC '08, Springer-Verlag, pp. 139–150.